MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report

of

Computer Networks LAB

TITLE: Implementation of Go-Back-N

SUBMITTED

BY

Jampula Sai Rahul Chowdhary – 200905088 (Roll no. 19)

Pranai Rao Marru – 200905091 (Roll no. 21)

1: INTRODUCTION

In this project, we have implemented, the two aforesaid reliable data transfer protocols i.e. GBN and SR over the unreliable transport layer protocol UDP. Mechanisms to demonstrate their robustness in events of errors such as checksum error, lost packets and ACKS's have also been deployed in the program, the implementation of which has been undertaken in C language.

1.1 General Introduction to the Topic

Go-Back-N (GBN)

In a **Go-Back-N (GBN) protocol**, also known as sliding window protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, *N*, of unacknowledged packets in the pipeline. Go-Back-N (GBN), has been implemented as an enhancement rdt (reliable data transfer 3.0) protocol. Allowing sender to transmit multiple packets without waiting for an acknowledgement, the former is constrained by the window size N which defines the maximum number of unacknowledged segments with each segment in the window being assigned a unique sequence number, in accordance to which an ACK shall be sent by the receiver. With a timer implemented at the sender, retransmission of the entire window shall be initiated upon the event of timeout. Figure 1.1 shows go back n in operation.

Selective Repeat (SR)

Selective-repeat protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error (that is, were lost or corrupted) at the receiver. This individual, as- needed, retransmission will require that the receiver *individually* acknowledge correctly received packets. A window size of *N* will again be used to limit the number of outstanding, unacknowledged packets in the pipeline. However, unlike GBN, the sender will have already received ACKs for some of the packets in the window. With timer implemented at sender, only the segment for which acknowledgement was not received shall resent in the event of a timeout, thereby reducing the Go-Back-N's need of large retransmissions. Figure 1.2 shows the SR operation and Figure 1.3 shows sender's view of the sequence number space.

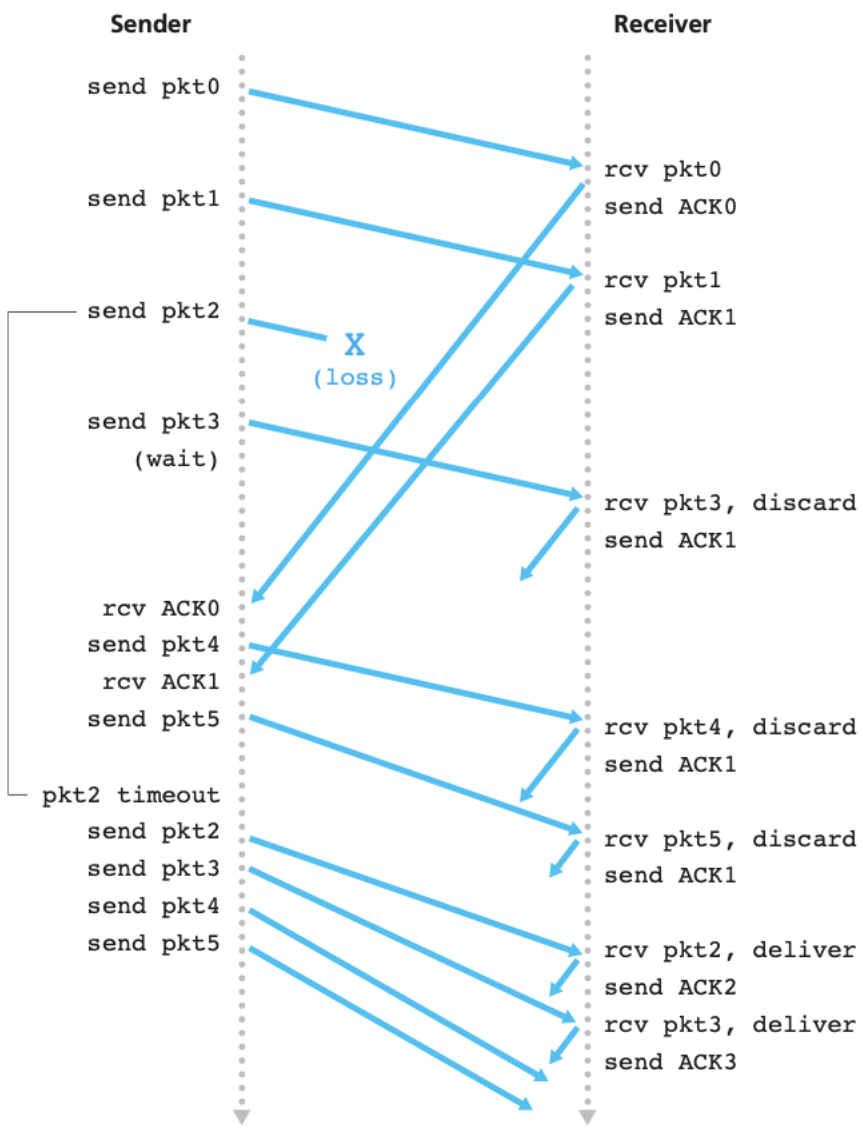


Figure 1.1: Go-Back-N in operation

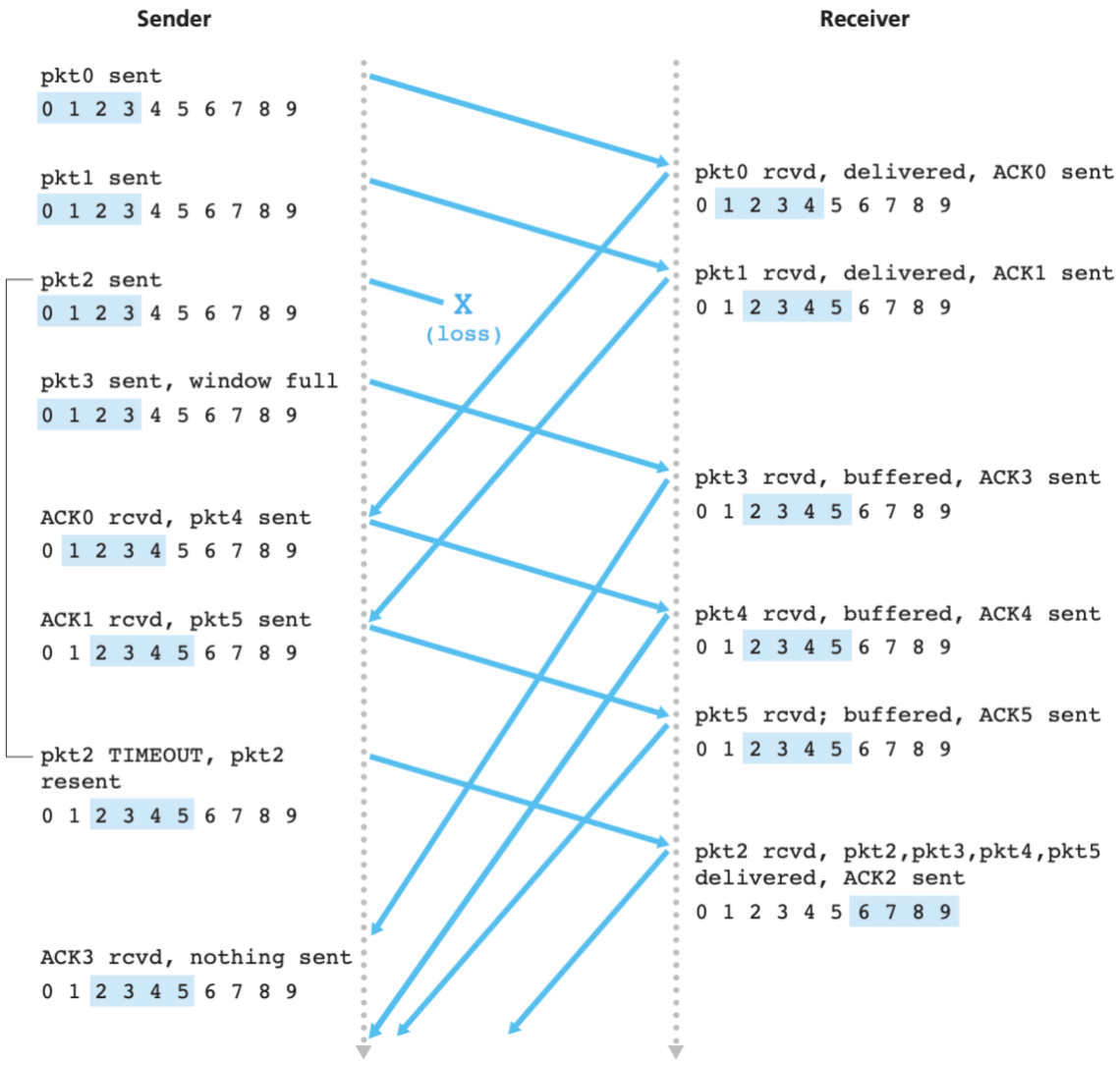


Figure 1.2: SR operation

SR sender events and actions

1. Data received from above. When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; other- wise it is either buffered or returned to the upper layer for later transmission, as in GBN.

2. Timeout. Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].

3. ACK received. If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to send base, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

SR receiver events and actions

1. Packet with sequence number in [rcv_base, rcv_base+N-1] is correctly received. In this case, the received packet falls within the receiver's win- dow and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (rcv_base in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with rcv_base) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of rcv_base=2 is received, it and packets 3, 4, and 5 can be delivered to the upper layer.

2. Packet with sequence number in [rcv_base-N, rcv_base-1] is cor- rectly received. In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.

3. Otherwise. Ignore the packet.

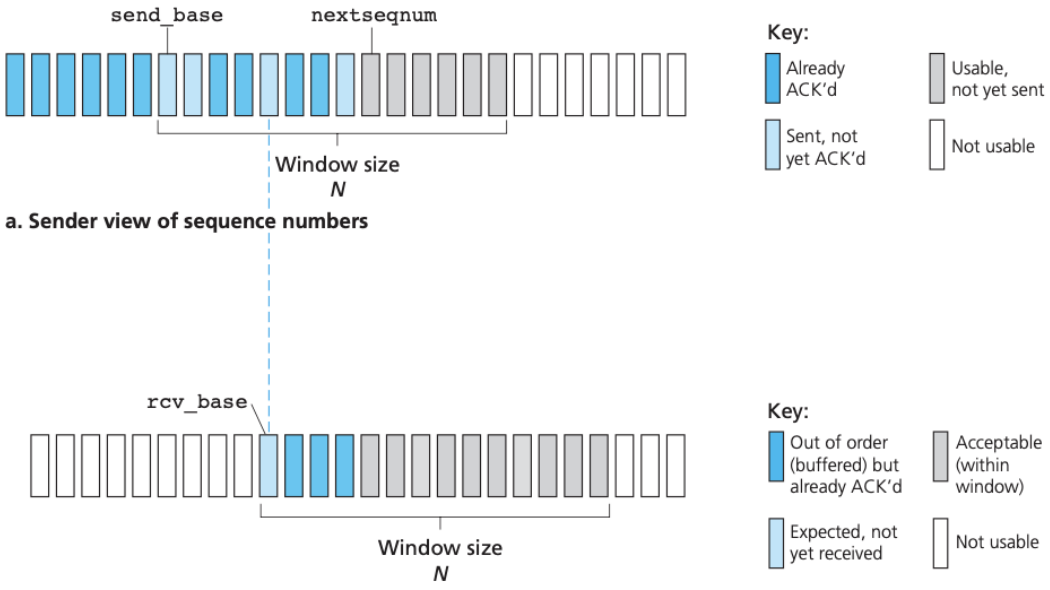


Figure 1.2: Selective-repeat (SR) sender and receiver views of sequence-number space

1.2 Hardware and Software Requirements

This project requires working pc with ubuntu software.

2: PROBLEM DEFINITION

This project is to implement two sliding window protocols – Go-Back-N (GBN) and Selective Repeat (SR) - using an unreliable channel (i.e. UDP). In order to implement these protocols you will need to be able to verify the contents of a transmitted package. Since UDP is an unreliable channel you will need to implement a checksum and append it to the front of the message before it's sent. The receiver will then need to use the checksum to verify the contents of the message. As expected, this project requires two Programs/ processes - sender process and receiver process.

In this project we used the Internet checksum to verify the contents of messages. The checksum is 16-bits long and works by summing up the message contents and then inverting the calculated sum. It's important to note that because the size of the sum is only 16-bits and it's important to add any values that would be carried out back into the sum. Sequence number is also included in the message in order to detect out of order packets.

3: OBJECTIVES

Our objective at sender program is that it should take the following command line arguments: input file, port number on which receiver is connected and number of packets you want to send. Using the parameters passed, the sender will communicate with the receiver process using UDP. During the session, the sender will display the following information for each segment sent and received:

1. The sequence number of the segment sent to the receiver and timer information.
2. Display the acknowledgement number received from the receiver.
3. When the timer for a particular sequence number expires, it should resend segments according to the protocol you are implementing. Display relevant information appropriately.

Our objective at receiver program is that it should take the following command line arguments: input file (so you can determine the type of protocol to use) and the port number the receiver will run on.

1. Create a socket with the specified port number and it will wait for client.
2. Display the sequence numbers that are received.
3. Send the acknowledgement according to protocol

4: METHODOLOGY

By the end of the project we plan to test the implementation and observe the robustness of the protocol by including some faulty network behaviours. The cases are

1. Bit error/Checksum error: Just before sending the segment you will intentionally alter/change some bits or bytes so your checksum will not work at the receiver. We want see the receiver detect the error with the packet. Assume that this bit error/checksum error has a probability of 0.1 to occur (i.e. roughly 10 segments out of 100 may have such issue).
2. Lost Packet: In this case we want see the response from receiver for a lost packet. The receiver will treat a particular packet as a lost packet even though it is actually arrived perfectly. Assume the lost packet has a probability of 0.1 (i.e. roughly 10 segments out of 100 may have such issue).
3. Lost ACK: Same as lost packet but happens at sender. We want see the response from sender for a lost ACK. Assume lost ACK has a probability of 0.05 (i.e. roughly 5 ACKs out of 100 will have such issue).

5: IMPLEMENTATION DETAILS

The application will be a command line interface (CLI), and we will need to pass an input file, which will contain the following inputs:

1. Protocol Name: GBN or SR
2. m N; where m = no of bits used in sequence numbers, N = Window size
3. Timeout period in micro second/millisecond. Use your judgement to set the value during simulation (4 seconds is

   good for testing).

4. Size of the segment in bytes

Example 1:

GBN
4 15
10000000
500

Example 2:

SR
4 8
10000000
500

Instructions for Execution of Program

1. Heading to the directory where the receiver program is stored from the terminal of Linux, execute the make file for the server, to compile the program.
2. Execute the z_exec_receiver file to start the receiver. $ ./exec_receiver inputfile 1234
3. On a new terminal window, heading to the directory of sender program, execute the make file to compile the server program.
4. Execute the z_exec_sender file to start the sender. $ ./exec_sender inputfile 1234 21

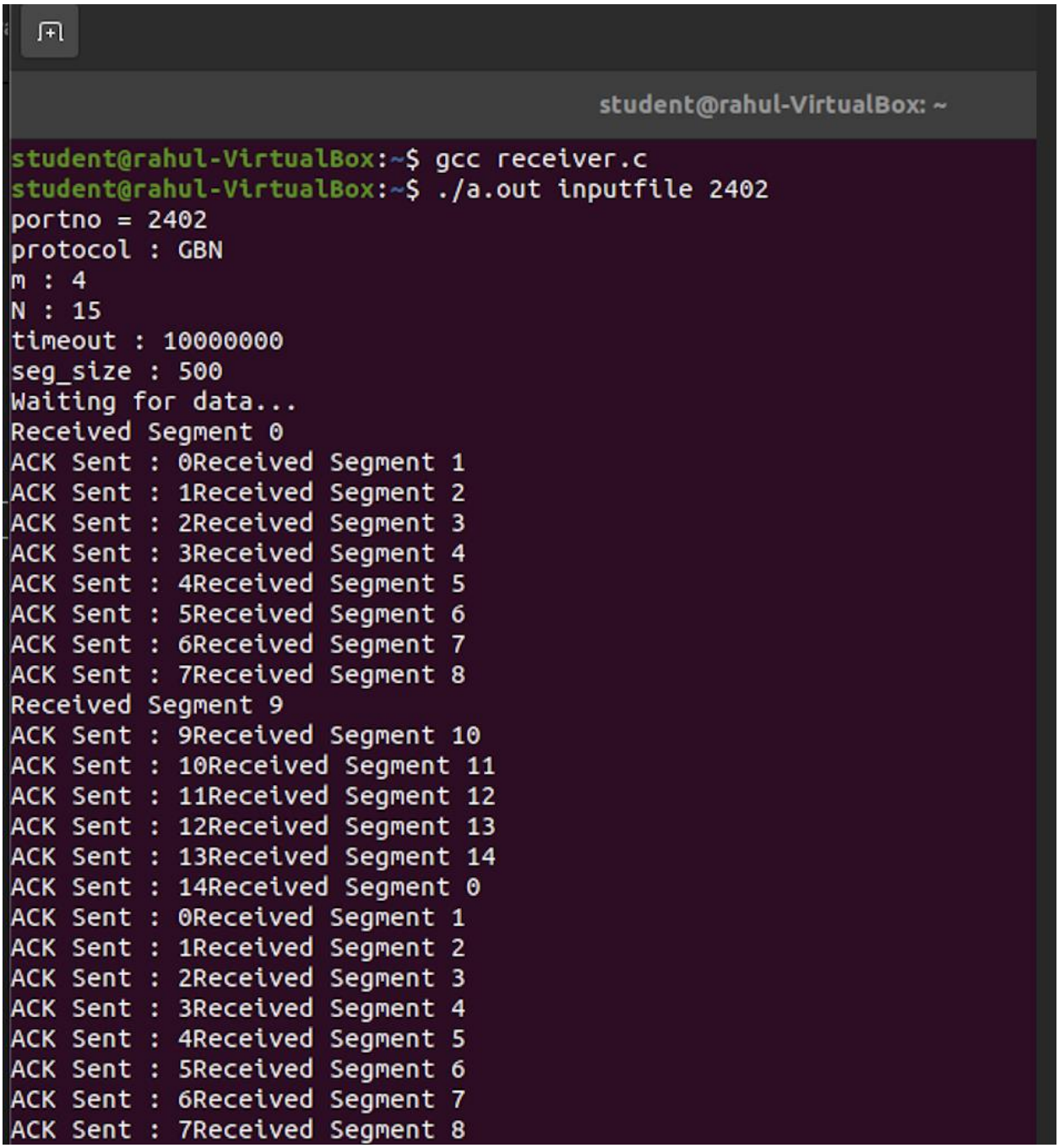
OUTPUT

GBN Sender Side

```
student@rahul-VirtualBox:~$ ./a.out inputfile 2402 30
portno = 2402
timeout : 10
no_of_packets = 30
seg_size : 500
protocol : GBN
m : 4
N : 15
sequence no range = 16
window size = 15

socket created

Sending 0; Timer Started
Received ACK : 0        Expected ACK : 0
Received ACK : 1        Expected ACK : 1
Received ACK : 2        Expected ACK : 2
Received ACK : 3        Expected ACK : 3
Received ACK : 4        Expected ACK : 4
Received ACK : 5        Expected ACK : 5
Received ACK : 6        Expected ACK : 6
Received ACK : 7        Expected ACK : 7
Received ACK : 9        Expected ACK : 8
Received ACK : 10        Expected ACK : 9
Received ACK : 11        Expected ACK : 10
Received ACK : 12        Expected ACK : 11
Received ACK : 13        Expected ACK : 12
Received ACK : 14        Expected ACK : 13
ACK not received recvfrom failed
Timer Expired; Resending the complete Window

Sending 0; Timer Started
Received ACK : 0        Expected ACK : 0
Received ACK : 1        Expected ACK : 1
Received ACK : 2        Expected ACK : 2
Received ACK : 3        Expected ACK : 3
Received ACK : 4        Expected ACK : 4
Received ACK : 5        Expected ACK : 5
Received ACK : 6        Expected ACK : 6
Received ACK : 7        Expected ACK : 7
Received ACK : 8        Expected ACK : 8
Received ACK : 9        Expected ACK : 9
```

```
Received ACK : 10        Expected ACK : 10
Received ACK : 11        Expected ACK : 11
Received ACK : 12        Expected ACK : 12
Received ACK : 13        Expected ACK : 13
Received ACK : 14        Expected ACK : 14
Communication successfull sliding window 15

Sending 15; Timer Started
Received ACK : 0        Expected ACK : 0
Received ACK : 1        Expected ACK : 1
Received ACK : 2        Expected ACK : 2
Received ACK : 3        Expected ACK : 3
Received ACK : 4        Expected ACK : 4
Received ACK : 5        Expected ACK : 5
Received ACK : 6        Expected ACK : 6
Received ACK : 7        Expected ACK : 7
Received ACK : 8        Expected ACK : 8
Received ACK : 9        Expected ACK : 9
Received ACK : 10        Expected ACK : 10
Received ACK : 11        Expected ACK : 11
Received ACK : 12        Expected ACK : 12
Received ACK : 13        Expected ACK : 13
Received ACK : 14        Expected ACK : 14
Communication successfull sliding window 30
```

GBN Receiver Side



```
ACK Sent : 8Received Segment 9
ACK Sent : 9Received Segment 10
ACK Sent : 10Received Segment 11
ACK Sent : 11Received Segment 12
ACK Sent : 12Received Segment 13
ACK Sent : 13Received Segment 14
ACK Sent : 14Received Segment 0
ACK Sent : 0Received Segment 1
ACK Sent : 1Received Segment 2
ACK Sent : 2Received Segment 3
ACK Sent : 3Received Segment 4
ACK Sent : 4Received Segment 5
ACK Sent : 5Received Segment 6
ACK Sent : 6Received Segment 7
ACK Sent : 7Received Segment 8
ACK Sent : 8Received Segment 9
ACK Sent : 9Received Segment 10
ACK Sent : 10Received Segment 11
ACK Sent : 11Received Segment 12
ACK Sent : 12Received Segment 13
ACK Sent : 13Received Segment 14
ACK Sent : 14
```

Selective Repeat Sender Side

```
student@rahul-VirtualBox:~$ ./a.out inputfile 2435 30
portno = 2435
timeout : 10
no_of_packets = 30
seg_size : 500
protocol : SR
m : 4
N : 15
sequence no range = 16
window size = 15

socket created

Sending 0; Timer Started
Received ACK : 0        Expected ACK : 0
Received ACK : 1        Expected ACK : 1
Received ACK : 2        Expected ACK : 2
Received ACK : 3        Expected ACK : 3
Received ACK : 4        Expected ACK : 4
Received ACK : 5        Expected ACK : 5
Received ACK : 6        Expected ACK : 6
Received ACK : 7        Expected ACK : 7
Received ACK : 9        Expected ACK : 8
Received ACK : 10       Expected ACK : 9
Received ACK : 11       Expected ACK : 10
Received ACK : 12       Expected ACK : 11
Received ACK : 13       Expected ACK : 12
Received ACK : 14       Expected ACK : 13
ACK not received recvfrom failed
Timer Expired; Resending the failed packets

Sending 0; Timer Started
Received ACK : 0        Expected ACK : 0
Communication successfull sliding window 15

Sending 15; Timer Started
Received ACK : 1        Expected ACK : 1
Received ACK : 2        Expected ACK : 2
Received ACK : 3        Expected ACK : 3
Received ACK : 4        Expected ACK : 4
Received ACK : 5        Expected ACK : 5
Received ACK : 6        Expected ACK : 6
Received ACK : 7        Expected ACK : 7
```

```
Received ACK : 9        Expected ACK : 9
Received ACK : 10       Expected ACK : 10
Received ACK : 11       Expected ACK : 11
Received ACK : 12       Expected ACK : 12
Received ACK : 13       Expected ACK : 13
Received ACK : 14       Expected ACK : 14
Received ACK : 0        Expected ACK : 0
Communication successfull sliding window 30
```

Selective Repeat Receiver Side

```
student@rahul-VirtualBox:~$ gcc receiver.c
student@rahul-VirtualBox:~$ ./a.out inputfile 2435
portno = 2435
protocol : SR
m : 4
N : 15
timeout : 10000000
seg_size : 500
Waiting for data...
Received Segment 0
ACK Sent : 0Received Segment 1
ACK Sent : 1Received Segment 2
ACK Sent : 2Received Segment 3
ACK Sent : 3Received Segment 4
ACK Sent : 4Received Segment 5
ACK Sent : 5Received Segment 6
ACK Sent : 6Received Segment 7
ACK Sent : 7Received Segment 8
Received Segment 9
ACK Sent : 9Received Segment 10
ACK Sent : 10Received Segment 11
ACK Sent : 11Received Segment 12
ACK Sent : 12Received Segment 13
ACK Sent : 13Received Segment 14
ACK Sent : 14Received Segment 0
ACK Sent : 0Received Segment 1
ACK Sent : 1Received Segment 2
ACK Sent : 2Received Segment 3
ACK Sent : 3Received Segment 4
ACK Sent : 4Received Segment 5
ACK Sent : 5Received Segment 6
ACK Sent : 6Received Segment 7
ACK Sent : 7Received Segment 8
ACK Sent : 8Received Segment 9
ACK Sent : 9Received Segment 10
ACK Sent : 10Received Segment 11
ACK Sent : 11Received Segment 12
ACK Sent : 12Received Segment 13
ACK Sent : 13Received Segment 14
ACK Sent : 14Received Segment 0
ACK Sent : 0^C
```

6: CONTRIBUTION DETAILS

Code related to GBN – Pranai

Code related to SR - Rahul

Design layout and Documentation – both

7:REFERENCES

1. James F. Kurose and Keith W. Ross COMPUTER NETWORKING: A Top-Down Approach